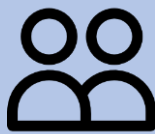


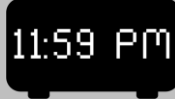
## Taller en Sala Nro. 3 Recursión Avanzada



En la vida real, las torres de Hanoi se utilizan para definir esquemas de rotación de backups <http://bit.ly/2hAqRkX>. También se utilizan como un examen neurológico para evaluar deficiencias en el lóbulo frontal. <http://bit.ly/2fYZmRL>



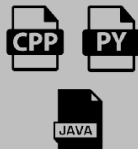
Trabajo en  
Parejas



Hoy, plazo  
máximo de  
entrega



Docente entrega  
código suelto en  
GitHub



Sí .cpp, .py  
o .java



No .zip, .txt,  
html o .doc



Alumnos  
entregan  
código suelto  
por GitHub

## Ejercicios a resolver

1. El problema de las torres de Hanoi es el siguiente: Tenemos 3 postes y  $n$  discos que colocar en los postes. Los discos difieren en tamaño e inicialmente están en uno de los postes en orden del más grande (disco  $n$ ) al más pequeño (disco 1).

La tarea es mover todos los discos a otro poste, pero obedeciendo las siguientes reglas:

- ☒ Sólo se puede mover un disco a la vez
- ☒ Nunca coloques un disco sobre uno más pequeño que él

Implementen un método que imprima en la pantalla los movimientos que hay que hacer para mover los discos de un poste a otro. **Paso 1:** condición de parada.

2. Escriban un programa que calcule las combinaciones de letras de una cadena. Una combinación es un subconjunto de los  $n$  elementos, independiente del orden. Existen  $2^n$  subconjuntos. Como un ejemplo, para “abc”, debe dar esta respuesta:

a ab abc ac b bc c

3. Escriban un programa que calcula las  $n!$  permutaciones de una cadena. Como un ejemplo, cuando uno le dicen “abc” debe dar la siguiente salida:

bca cba cab acb bac abc

4. **[Ejercicio Opcional]** Calculen las permutaciones de una cadena, como en el ejercicio 3, pero permitiendo que las letras se repitan, por ejemplo, permutaciones nuevas que aparecerían son aaa, aab, bcc, etc.

5. **[Ejercicio Opcional]** Implementen la funcionalidad del tarrito de paint. Representen el lienzo de paint como una matriz de enteros.

# Ayudas para resolver los Ejercicios

**Ayudas para el Ejercicio 1.....** [Pág. 4](#)

**Ayudas para el Ejercicio 2.....** [Pág. 4](#)

**Ayudas para el Ejercicio 3.....** [Pág. 5](#)

## Ayudas para el Ejercicio 1



**Pista 1:** Definan la función de esta forma:

```
private static void torresDeHanoiAux(int n, int origen, int
intermedio, int destino) {

}

public static void torresDeHanoi(int n) {
    torresDeHanoiAux(n, 1, 2, 3);
}
```



**Pista 2:** [https://www.youtube.com/watch?v=5\\_6nsViVM00](https://www.youtube.com/watch?v=5_6nsViVM00)

## Ayudas para el Ejercicio 2



**Pista 1:** Consideren el siguiente código:

```
public static void combinations(String s) {
    combinationsAux("", s);
}
```



**Pista 2:** Primero definan una función auxiliar de esta forma:

```
private static void combinationsAux(String base, String s)
{ ...
}
```



**Pista 3:** Si tienen dudas sobre cuáles son los subconjuntos de un conjunto, vean este video <https://www.youtube.com/watch?v=Bxv2Kvlltxs>



**Pista 4:** No se preocupen en qué orden obtiene las combinaciones.



**Pista 5:** La función recursiva debe generar el siguiente árbol de ejecución para generar los subconjuntos de la cadena “abc”.



**Error Común:** Un error común es calcular los prefijos en lugar de los subconjuntos, como se muestra en el siguiente ejemplo. La solución es hacer 2 llamados recursivos y no uno solo.

```
public static void prefijos(String base, String s){
    if (s.length() == 0){
        System.out.println(base);
    }
    else {
        prefijos (base + s.charAt(0), s.substring(1));
        System.out.println(base);
    }
}
```

## Ayudas para el Ejercicio 3




**Pista 1:** Consideren el siguiente código:

```
private static void permutationsAux(String base, String s) {

}

public static void permutations(String s) {
    permutations("", s);
}
```

|   |  |                        |
|---|--|------------------------|
|  | UNIVERSIDAD EAFIT<br>ESCUELA DE INGENIERÍA<br>DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS | Cód. ST0245            |
|   |  | Estructuras de Datos 1 |



**Pista 2:** No se preocupen en qué orden obtiene las permutaciones.



**Pista 3:** La función recursiva debe generar el siguiente árbol de ejecución para generar las permutaciones de longitud 3 de la cadena “abc”.

# ¿Alguna inquietud?

## CONTACTO

**Docente Mauricio Toro Bermúdez**

**Teléfono:** (+57) (4) 261 95 00 **Ext.** 9473

**Correo:** mtorobe@eafit.edu.co

**Oficina:** 19- 627

Agende una cita con él a través de **<http://bit.ly/2gzVg10>** , en la pestaña *Semana*. Si no da clic en esta pestaña, parecerá que toda la agenda estará ocupada.